White-Box Validation of Collective Adaptive Systems by Statistical Model Checking and Process Mining

Roberto Casaluce[1], Max Tschaikowski[2], and Andrea Vandin[1,3(✉)]

[1] Sant'Anna School for Advanced Studies Pisa, Pisa, Italy
andrea.vandin@santannapisa.it
[2] Aalborg University, Aalborg, Denmark
[3] DTU Technical University of Denmark, Kongens Lyngby, Denmark

Abstract. Modeling and analyzing collective adaptive systems with heterogeneous components poses challenges to language designers, software engineers, and computer scientists interested in the formal verification of the modeled systems. This requires the integration of computational, reasoning, and analysis tools, but also of techniques to validate and shed light on the actual behavior described by a model. We build on a methodology built in collaboration with Rocco De Nicola, based on his influential contributions in the modeling and programming of distributed and adaptive systems. Such methodology allows to model and analyze collective adaptive systems equipped with reasoning capabilities. It combines SCEL, a language for programming collective adaptive systems, with PIRLO, a reasoner that can enrich agents with reasoning capabilities, and MultiVeStA, a statistical model checker that can analyze the obtained models by simulating them. Here, we further enrich this methodology with techniques from process-oriented data science, and in particular process mining, to ease the validation and debugging of such models. This follows recent proposals from the literature that validate models by explaining graphically the behaviors observed by a statistical model checker. We demonstrate our approach by considering a simple collision-avoidance robotic scenario where a group of robots moves in an arena while aiming at minimizing the number of collisions.

Keywords: SCEL · Statistical Model Checking · Process mining

1 Introduction

Collective Adaptive Systems (CAS) represent a complex and dynamic field of study in computer science and other disciplines [2,16,17,26,30,38,39,43]. These systems, characterized by the integration and interaction of many heterogeneous components, pose significant challenges for language designers, software engineers, and computer scientists interested in analyzing the modeled systems. The validation and formal verification of such systems necessitates a comprehensive

T. Margaria and B. Steffen (Eds.): ISoLA 2024, LNCS 15219, pp. 204–222, 2025.
https://doi.org/10.1007/978-3-031-73709-1_13

approach that combines computational, reasoning, and analysis tools. In this paper, we base on a methodology presented in [8] that allows to model and analyze CAS with reasoning capabilities. The methodology in [8] leverages the synergy between SCEL [34,35] —a language tailored for programming CAS—, Pirlo [7], a reasoner that enriches agents with decision-making capabilities, and MultiVeStA [24,37,42], a statistical model checker (SMC) to study the behavior of these systems by simulation-based techniques equipped with statistical guarantees. Such methodology is based on high-impact proposals of Rocco De Nicola on languages for the modeling, programming, and analysis of distributed and collective adaptive systems (e.g., [3,14,31–33,35]).

Here, we further this methodology by incorporating recent proposals that integrate SMC with process mining (PM) techniques [18] to validate and debug the models, providing a clearer understanding of the actual behavior that they describe. PM belongs to the family of process-oriented data science. It encompasses a number of techniques aimed at using data to study, mine, and enhance the process that generated them [1] (the data-generation process). In particular, PM techniques divide in: process identification, which performs reverse-engineering of the data to mine a graphical description of the process, process enhancement, which proposes improvements to a process, conformance checking, which checks whether and how a process deviates from another one. Our methodology prioritizes process identification and variants of conformance checking.

In [18], we propose a graphical *diff model* which highlights the differences among the mined and expected (reference) processes. It is given in [18] in the same graphical model specification language of the input model. This has the advantage of not requiring the use of a new formalism. On the other hand, this limits the approach to languages with graphical components. Here, we propose a novel notion of diff model that makes the formalism independent of graphical representations. We demonstrate our methodology using a collision-avoidance robotic scenario implemented in SCEL and Pirlo from [8]. Thanks to our methodology, we discover and fix unexpected behaviors in the model due to subtle issues in the code integrating SCEL and Pirlo. We therefore demonstrate how our approach can advance the modeling, validation, and debugging of CAS models offering insights into their intricate dynamics, shedding light in complex contexts where components are equipped with reasoning capabilities. We consider the Maude-based implementations of SCEL and Pirlo from [8], MISSCEL, which is integrated with MultiVeStA for the analysis tasks.

The paper is structured as follows: Sect. 2 presents the robotic scenario. Section 3 provides background material, while Sect. 4 introduces our recent approach combining SMC and PM, discussing how it can be applied to the domain considered here. Finally, Sect. 5 applies the enriched methodology to the robotic scenario, while Sect. 6 concludes the paper.

2 A Collision Avoidance Robotics Scenario

We consider the simple scenario from [8]: a group of robots moving in an arena. Some robots, the *normal* ones, are random walkers, while others, the *informed*

ones, attempt to minimize the collisions by feeding an internal reasoner with information on the number of robots perceived in the neighboring positions.

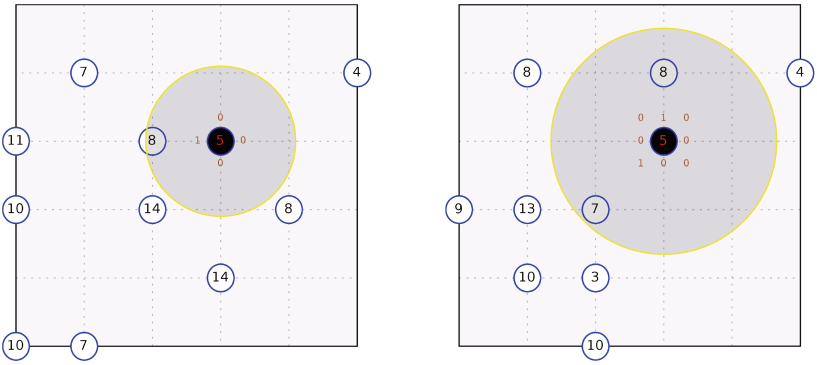

Fig. 1. A graphical representation of the scenario with random walkers (white circles) and informed robots (black circle) counting collisions (the numbers in the circles). The robots perceived by informed ones are given, for each direction, by the small numbers surrounding them. Left: the informed robot perceives robots in the up, left, down, right neighboring positions. Right: the informed robot also perceives robots in the diagonal positions.

Two variants of the scenario are depicted in Fig. 1. The arena is abstracted as a discrete grid (the gray dashed lines), with robots (the white or black circles) being situated in cells intersections. Multiple robots may be in the same position, in which case they *collide*. The number inside each robot denotes the number of collisions it had so far. Robots move one cell at a time following a dashed line (up, right, down or left), or stay idle. Normal robots (white circles) randomly choose the action to be executed among the five listed above, i.e., the direction or to stay idle. Informed robots, instead, perceive robots in their surrounding environment by relying on proximity sensors, and use this information to choose an action that minimizes the probability of collision with others. The portion of environment perceived by informed robots depends on their perception range (the semi-transparent circles around informed robots in Fig. 1). For example, the informed robot of Fig. 1 (left) perceives its four neighboring positions, while the one of Fig. 1 (right) also perceives the four diagonal ones. The (four or eight) small numbers surrounding informed robots denote their perception of the environment. The informed robot in Fig. 1 (left) perceives 1 robot on its left, and 0 in the other directions. The informed robot in Fig. 1 (right) perceives 1 robot on its bottom-left and above it, and 0 in the other directions. Focusing on the informed robot in Fig. 1 (right), we note that the positions up, right, down and left are reachable in one step, while the diagonal ones in two. However, the information on the diagonal neighboring positions is be useful also for reasoning about the next step, because, e.g., the robot perceived in down-left could move

towards the same position chosen by the informed robot (e.g., up, if the informed robot moves left), leading to a collision.

3 Background

We now recall notions from SCEL (Sect. 3.1) and process mining (Sect. 3.2) necessary in the rest of the paper.

3.1 SCEL

The formal language SCEL is a process calculus introduced in [34] to model, program, and analyze collective adaptive systems made of interacting SCEL components. It is a highly general and parametric language. Here we focus on the SCEL_{TS} fragment [8], which is inspired by the popular language Klaim [12, 32, 33] based on tuple-spaces.

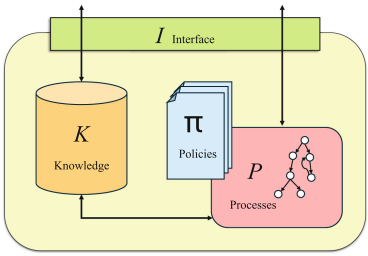

Fig. 2. SCEL component

SCEL offers *attribute-based communication* to identify groups of components with whom sharing (local) knowledge in the form of tuples, modeling complex interactions and coordination. It combines programming abstractions to precisely handle aggregations (the interactions among various components to create ensembles and systems), behaviors (the advancement of components), and knowledge manipulation, all guided by specific policies. As depicted in Fig. 2, a SCEL component consists of a knowledge repository, an interface, a set of policies, and a process. In SCEL_{TS}, the knowledge repository is a set of tuples which can be seen as the set of facts known by the component, while the interface lists the segments of local knowledge *published*, i.e., accessible to other components. Policies oversee interactions within the internal elements of a component (interaction policy) and with other components (authorization predicate). In SCEL_{TS} we only have default policies: within components, processes evolve in a pure interleaving fashion, while interactions among components are always authorized. Lastly, a process is utilized to specify the behavior of the component. Processes carry out local computations, coordinate interactions with local knowledge, or engage with remote repositories (following the restrictions imposed by the interfaces and policies of the involved components). Specifically, three actions are available to interact with repositories: **put**, **qry**, and **get**, which correspond to adding, retrieving, and withdrawing, respectively, tuples. These actions can have as target either **self**, the own knowledge of the component, or a target predicate. Target predicates allow to interact with other components in a point-to-point manner (by providing the identifier of a specific component), or to conduct attribute-based communication with all (**put**) or any (**qry** or **get**) component that satisfy the target predicate.

MISSCEL. In [8], an implementation of SCEL in Maude [21], MISSCEL, has been presented. MISSCEL provides an implementation of SCEL in rewriting logic. Thanks to MISSCEL, a SCEL specification can be analyzed using the extensive tool-set offered by Maude [21], such as state-space generation, debugging, LTL model checking, or statistical model checking using Multi-VeStA [37,42].

```
1  SC(I( tId('SCId)),
2     K( < tId('SCId) ; av(id('robot-normal-1)) >, < tId('type) ; av('normal)
           >,
3        < tId('pos) ; av(2) av(2) >, < tId('collisions) ; av(8) >),
4     Pi(INTERLEAVING-INTERACTION-PREDICATE),
5     P( qry(< tId('pos) ; ?x('x) ?x('y) >)@ self .
6        put(< tId('dir) ; randomDirection(x('x), x('y)) >)@ self .
7        put(< av('terminated) >)@ self [ get(< av('terminated) >)@ self .
8                                         pDef('PnormalRobot)] )
```

Listing 1. A (MIS)SCEL component representing a normal robot

Listing 1 shows the MISSCEL representation of a SCEL component modeling a normal robot. It is a Maude term of type `ServiceComponent`, with constructor

```
op SC: Interface Knowledge Policies Processes -> ServiceComponent.
```

The interface states that only the ID of the robot is published (line 1). Lines 2–3 contain the knowledge of the robot, namely the ID (`'robot-normal-1`), the robot type (`'normal`), as well as the current position (`2,2`) and collision count (`8`). Line 4 specifies the default policy of SCEL_{TS}. Lines 5–8 contain the process of the robot which states that: the robot first retrieves its position from the local knowledge, then adds to its knowledge a randomly chosen direction (`randomDirection`), which depends on the current one (the robot cannot move beyond the boundaries of the arena). The term `randomDirection` is transformed in an actual direction, randomly chosen among the available ones for the robot, by Maude equations not shown here. The Maude equation presented in Listing 2 utilizes this direction to actually perform a movement. The function `computeNeighboringPosition` updates the `x` and `y` coordinates based on the specified direction (`dir`). Subsequently, line 7 of Listing 1 adds the token `terminated` to the local knowledge to indicate the completion of the movement. This token can then be consumed by the process enclosed in square brackets (running concurrently with the previously described process), which causes the robot to iterate its behavior from Line 5 (we invoke the process definition `pDef('PnormalRobot)` which contains the process of the normal robot).

We refer to [8] for more information on the semantics of SCEL and of its implementation in MISSCEL.

Pirlo. The reasoner used by informed robots in [8] to minimize the number of collisions is Pirlo [7], an implementation of action programming in Maude [7].

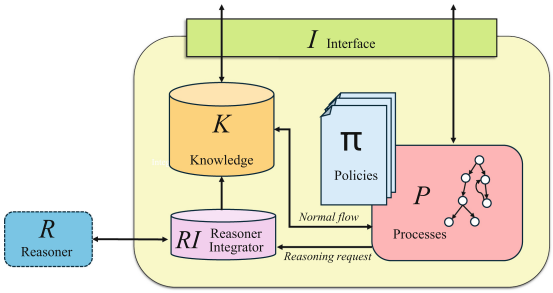

Fig. 3. Enriched SCEL component

From a language and software engineering perspective, the integration has been obtained by extending the architecture of SCEL components as shown in Fig. 3. Essentially, PIRLO is triggered by the addition of a special tuple (a reasoning request) in the knowledge of a component. Using ad hoc MAUDE equations, PIRLO processes the request and replaces it with the result (in the considered scenario, the direction to take). Further details can be found in [8].

The idea behind PIRLO is to write a non-deterministic action program [7] that captures all alternative behaviors of the agents (SCEL components, that is robots in the considered scenario). PIRLO processes such action program to compute the effects of these alternatives, e.g., by computing the probability of avoiding collisions when moving in a specific direction.

PIRLO needs to receive as an input the representation of the current *state* perceived by a robot. This representation is provided in terms of *fluents*, properties of the state that might change upon the execution of actions by agents. Furthermore, the *dynamics of the domain*, i.e., the effects of actions on fluents have to be specified. This forms a *background knowledge* for the reasoner that can be used to predict the effects of the execution of actions. More information on this can be found in [8]. Lines 1–2 of Listing 3 show that the sort `Fluent` is subsort of `State`, i.e., a fluent term is also a state term, while line 3 shows that a complex state can be specified as the conjunction of simpler state terms. Lines 5–9 show that fluents can be created using operators `p` (probability of being in a certain state), and `pos` (the position of a robot). For example, `pos(a1,2,3) and pos(a2,2,3) and p(0.8)` denotes a state with agents `a1` and `a2` in positions `2,3`, with the additional property that this state occurs with a

```
1  ceq SC(I, K(< tId('pos); av(x)  av(y)  >, < tId('dir) ; av(dir) >, k), Pi,
       P)
2    = SC(I, K(< tId('pos); av(x2) av(y2) >                          , k), Pi,
       P)
3   if av(x2) av(y2) := computeNeighboringPosition(av(x),av(y),av(dir)).
```

Listing 2. Maude equation for robot movement execution

```
sorts State Fluent
subsort Fluent < State
op _and_ : State State -> State [assoc comm ...] .

sorts Agent  Position  Probability.
subsort Probability < Fluent .
op p : Float -> Probability.
subsort Position < Fluent .
op pos : Agent Int Int -> Position.
```

Listing 3. Sorts and operators to exemplify PIRLO's state

```
var A : Agent .
vars X, Y, DX, DY : Int .

crl pos(A, X     , Y     ) and move(A, DX, DY)
=>  pos(A, X + DX, Y + DY)
if X + DX and Y + DY are in grid area .
```

Listing 4. Sketch of rewriting rule for specifying domain dynamics in PIRLO (effects of the `move` action)

probability of 0.8. All this machinery has been implemented in the integration of MISSCEL and PIRLO in [8] to which we refer for further details.

As discussed, it is necessary to specify the domain dynamics. These are given as rewrite rules. From the perspective of PIRLO, in the robotic scenario agents only have one action, `move`, which models the fact that a robot changes position (e.g., `move(a1,1,0)` denotes that agent `a1` will move one step right). Listing 4 exemplifies the effects of this action on the fluent `pos` of the considered agent. This rule describes how a movement changes the fluent `pos`. To compute the probability of colliding, the informed robot is provided by PIRLO with the probability of perceiving the future outcome after each action performed by the other agents (the `p` terms in Listing 3). Each other agent is considered a normal robot, i.e., a random walker. In other words, it assumes that the normal robots randomly choose an action between staying idle or moving in one of the four directions with uniform probability (each with probability of 0.2).

MultiVeStA. The analysis tasks are performed with MultiVeStA, a black-box SMC (Statistical Model Checking) tool. MultiVeStA has been originally proposed in [37] to study transient properties (i.e., at given time points) of discrete-event simulators. Recently, it has been redesigned to target agent-based models [42], providing it with richer algorithms for steady-state properties (i.e., properties evaluated after the system reaches an equilibrium), and for counterfactual analysis (i.e., what is the impact of a change in the parameters in the system dynamics?). Furthermore, as discussed, MultiVeStA has been recently integrated in [18] with process mining techniques. Similarly to other black-box statistical model checkers, MultiVeStA does not make assumptions regarding the studied model. It only assumes that it can be reset providing a new random seed, that single simulation steps can be performed, and that observations on the current

simulation state can be evaluated. This tool enables the analysis of complex scenarios and facilitate coping with the stochastic nature of probabilistic systems without exhaustively exploring system state spaces. It controls the simulator at hand to conduct n independent simulations, with n being sufficiently large, but minimal, to statistically estimate quantitative properties. The quantitative properties are evaluated via estimations provided with statistical guarantees. We refer to [42] for details on this. MultiVeStA has been successfully applied to a variety of domains, such as economic agent-based models [41,42], crowd steering scenarios [36], public transportation systems [20,25], lending pools in decentralized finance [4], highly-configurable systems [6,40], business process modeling [22], security threat modeling [5], self-assembly robotic scenarios [15], and further frameworks for collective adaptive systems [23]. In [8], MultiVeStA has been recently extended with logging capabilities [18]. With this extension, it is now possible to apply process mining techniques to post-process the simulations used to compute the SMC results, de facto explaining the obtained results. This enables us to conduct *white-box* validation and debugging of the model at hand. MultiVeStA has been applied in [8] to the considered robotic scenarios with planning capabilities using MISSCEL. In this paper we extend such analysis using process mining.

3.2 Process Mining

Process Mining (PM) is an interdisciplinary field that seeks to derive insights on a data-generation process by analyzing its executions, the *logs*, thereby bridging the gap between data science [27] and process science [1]. The main activities of PM encompass process discovery, process enhancement, and conformance checking. Discovery entails identifying a (ideally abstract and compact) representation of the executed process by consolidating all observed instances into one model. A model discovered can be enhanced with supplementary information, such as the frequency of executed activities or paths, or by improving its performance (e.g., by identifying and eliminating bottlenecks). Conformance checking, instead, allows to evaluate the degree of deviation between a *reference* model, with actual executions. In this paper, we focus on the discovery and (a variant of) conformance checking tasks of PM. Our goal is to synthesize models from logs generated during SMC analyses capturing the observed behavior in the simulations that led to the SMC results. We use an established process discovery algorithm, the Heuristics Miner (HM) algorithm [44] as implemented in pm4py [11], which facilitates the comparison of generated and expected behavior by offering a precise understanding of complex process dependencies. The HM algorithm constructs a Directly Follows Graph (DFG) from event logs, identifying *direct follow* relations between activities. It then appends start and end activities to each *case ID* (i.e., the activities of each simulation). Additionally, users have the flexibility to adjust parameters such as noise and dependency thresholds to balance model accuracy and the inclusion of less common paths. By fine-tuning these parameters, the likelihood of incorporating rare paths into the discovered process model increases, aiding in the identification of possibly uncommon rare

unwanted behaviors. As regards conformance checking, we implement our own variant of it as a *diff model* detailed later which highlights the differences in processes mined from models at different stages of maturity and debugging.

4 Recent Developments on Integrating Statistical Model Checking and Process Mining

In our recent work [18,19], we have proposed a novel method that augments SMC techniques with PM methods to *explain* graphically the results computed by MultiVeStA. In [18] we have shown how the integration of SMC and PM helps in identifying undesirable behaviors in formal models by visualizing the behavior expressed during the event logs of the required simulations. However, our method goes beyond simply applying discovery algorithms to event logs. Rather, we integrate SMC and PM techniques using a graphical component, the *diff model*, given in the model specification language. In [18] we considered the formal languages QFLan [6,40] and RisQFLan [5], but it can be generalized to other model specification languages. Our methodology takes SMC logs as input to post-process them, performs analysis tasks using PM techniques, and then visualizes the results using the model specification language. This approach aids the automatic discovery of missing or undesirable behavior within the model. Figure 4, adapted from [18], depicts the workflow of the SMC-PM analysis pipeline instantiated on the QFLan formalism:

1. *Model creation*: The methodology starts with creating a model. The modeler defines all system components using QFLan tools, which offer the automated generation of a graphical representation of the procedural part of the model;
2. *Logs generation*: The second step involves generating logs by simulating the model during SMC analysis of MultiVeStA;
3. *Logs pre-processing*: After creating the logs, they undergo necessary preprocessing before being able to apply PM techniques. This step helps prevent the loss of important information about the executed process;
4. *Process mining*: The heuristic miner algorithm is used to mine the preprocessed logs. The mined PM model is transformed into a *mined QFLan model*, able to express the behavior observed in the simulations;
5. *Automatic diff*: The graphical representations of both the original and mined QFLan models are parsed to create a diff model which highlights the differences between the two models. The diff model includes common edges and nodes in both models, colored in black, while edges and nodes unique to one model are highlighted in red. Dashed red edges indicate transitions present in the original model but missing in the mined one, whereas transitions with continuous red edges represent transitions present only in the mined model.

In [18], we demonstrated the effectiveness and scalability of our approach focusing on the formalisms QFLan [6,40] and RisQFLan [5], coming from the software engineering (product lines engineering) and security domains, respectively. A crucial common aspect of these two languages is that they come with

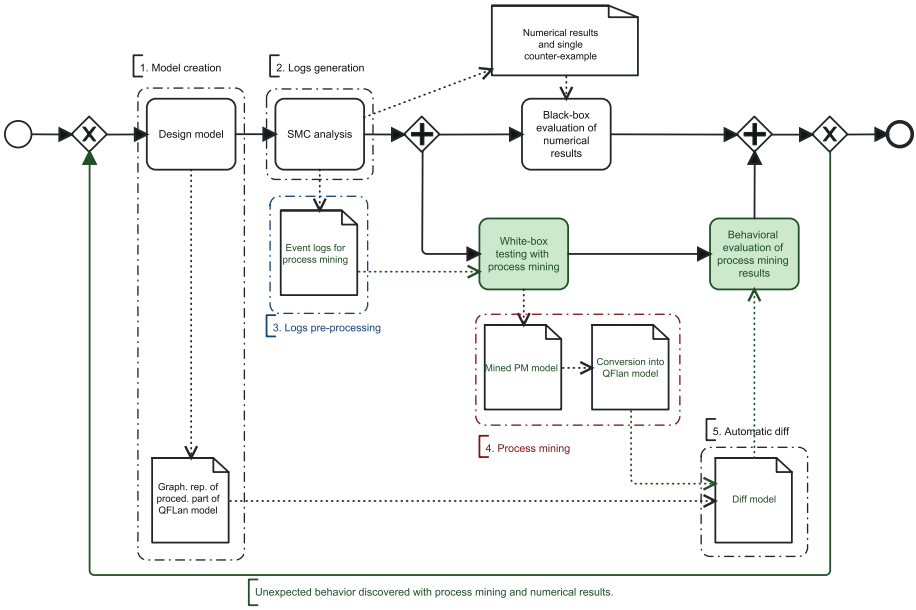

Fig. 4. Adapted from [18]: automatic white-box behavioral validation of QFLan models combining SMC with PM.

a graphical representation of their procedural part which is automatically generated by the underlying tool support. Our approach is heavily based on this. In this paper, we evaluate the application of this methodology to SCEL models enriched with reasoning capabilities, which do not feature graphical representations. Indeed, in this paper we demonstrate that our approach can be applied also to formalisms and simulators that do not generate a graphical representation of the procedural part of the formal model. Despite such a limitation, in Sect. 5 we apply our approach. We use a slightly different setting: we build a diff model highlighting the differences between two (batches of simulations of two) versions of a model at different stages of maturity. Essentially, this allows users to identify and analyze the changes made to the model during the debugging process. In the experiments in Sect. 5, we demonstrate how our diff model can identify differences between the original model exhibiting unwanted behaviors and the refined version without such behaviors.

5 Methodology Applied on the Scenario

5.1 SMC Analysis Results

In [8], we analyzed the robotic scenario using SMC. We evaluated the average number of collisions among the robots in the first 5000 steps of simulation using MultiVeStA. A step does not correspond to a movement of a robot, but

to the execution of one of the actions of the process of a SCEL component (see, the discussion in Listing 1). In order to have a probabilistic fair simulator for all robots, simulations proceed as follow: we randomly sort the IDs of robots, and perform a step per robot in the given order. In particular, as discussed in [8], for each robot we compute all admissible one-steps executions (i.e., the execution of a SCEL action), and choose randomly one of them with uniform probability. Once all robots have been considered, the IDs are sorted again randomly, and one step per each is performed following the new order, and so on. If a robot does not have any one-step execution, the system does not evolve in the step of such robot, and a special activity *deadlock* is added to the logs.

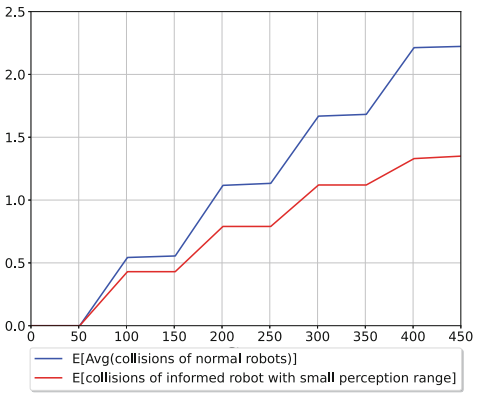

Fig. 5. Collisions in first 450 steps. Robotic scenario where the informed robot perceived 4 neighboring positions.

In [8], we have considered two scenarios by varying the perception range of the informed robot. One with smaller and one with larger perception range (see the discussion in Fig. 1). In [8] we have shown that, as expected, the average number of collisions for the normal robots is consistently higher than those of the informed ones. Furthermore, a wider perception range led to fewer collisions.

Here, we consider a simpler setting, further analyzed in the next section with PM. Figure 5 shows the SMC results for the scenario with smaller perception range for the first 450 simulation steps. Furthermore, to ease presentation, we imposed 100 as maximum number of simulations performed. For the informed robot, we provide the expected number of collisions at steps 50, 100, 150, 200, 250, 300, 350, 400, and 450. Instead, for the normal robots we provide the expected number of the average number of collisions of all normal robots (measured at the same points). In the next section, we demonstrate how we can use the logs generated during these simulations for validating and debugging the model.

5.2 Identifying and Solving Issues in the Model with PM

The logs generated by SMC are used to extract a process model, in particular a DFG model (see Sect. 3.2), describing the behavior of both types of robots (normal and informed). Visually inspecting a DFG can aid in identifying whether the behavior intended for a model is violated in the actual executions. Indeed, as discussed, we use PM techniques to explain black-box SMC analyses.

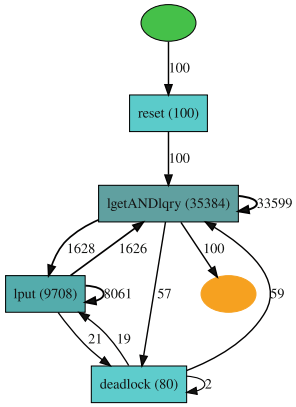

Fig. 6. Process model with informed and normal robots

Figure 6 illustrates a process model generated using the HM algorithm on the logs collected for the SMC analysis in Fig. 5. In this case, the HM algorithm has been applied without the logs undergoing any pre-processing. The green and orange circles correspond, respectively, to the start and end of each simulation added by HM before the first event and after the last event of each simulation. All other nodes are *activities* observed in the logs (the action executed by the processes of SCEL components). In the logs, we also have the ID of the robot that executed in that step, but we ignore it for the analysis on Fig. 6. The labels on the edges between the activities represent the overall number of times that the source activity is directly followed by the target one. For example, `reset`, a special activity added by MultiVeStA when resetting the simulator before each new simulation, is always followed (once) by an `lgetANDlqry` activity corresponding to the first action that is executed by SCEL components in our scenario. This denotes the execution of an action **`qry`** to access values in the local knowledge of the corresponding SCEL component (`self`), see discussion on Listing 1. We note a loop on `lgetANDlqry`, describing that all SCEL components need to perform a `qry`. From this state, we also have a transition toward `lput`. Indeed, the process of SCEL robots involves a `put` to `self` after the `qry` (Listing 1).

Moreover, Fig. 6 highlights issues in the behavior: it contains an unexpected `deadlock` activity. As discussed in Sect. 5.1, it means that while performing the simulation steps, MultiVeStA encountered robots that did not have any action to

perform, i.e., an unexpected behavior. In fact, normal robots just keep iterating on the process in Listing 1. The process of informed robots proceeds similarly, with extra actions necessary to interact with the reasoner. Therefore, the model as provided in [8] needs to be fixed to get rid of these unexpected deadlocks. As anticipated, we have used PM to discover the presence of this. We next demonstrate that PM can be used to shed further light on this aspect.

To fix the deadlock, we need first to identify which type of robot has exhibited the undesirable behavior. To this end, we selected only those actions in the logs connected to the informed robot, as well as to a randomly selected normal one. We then separated the events in two distinct logs, one for the informed robot, and one for the normal robot. Finally, we used these separate logs to mine two separate process models. The rationale behind choosing only one normal robot lies in the fact that all robots of the same type should display the same behavior, and if one exhibits unwanted behavior, the same should occur with the others.

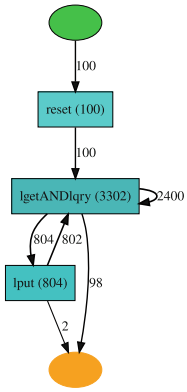

Fig. 7. DFG of a normal robot

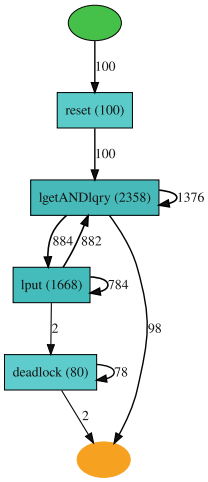

Fig. 8. DFG of the informed robot

Figure 7 depicts the DFG of the randomly selected normal robot from the logs. The mined process model shows only the expected activities that describe the execution flow of normal robots. Figure 8 shows the DFG mined using the logs only of the informed robot. In this model, we can identify a deadlock activity. In particular, the label `2` on the edge entering the `deadlock` activity tells us that this happened in two simulations. Notably, when generating the logs, MultiVeStA assigns to the events of a simulation the random seed of the simulation as *case ID*. Thanks to this, we were able to debug the two simulations leading to a deadlock, identifying the cause of the deadlock. We discovered that the issue was in the interaction with Pirlo. In particular, it turned out that, in these

two simulations, PIRLO did not choose a direction to take. Rather, it left a sort of *error value*, which prevented the MAUDE system from further applying rewriting rules (the MAUDE systems works by pattern matching, and the error values where unexpected and prevented matching any rule).

```
1  eq invokeReasoner(reasoningRequest(S, P)) =
2  reasoningResult(maxProb(metaExec(S, P))) .
3  ...
4  eq reasoner2scel(reasoningResult(A ; P)) = av(translateAction(A)) .
5  eq reasoner2scel(reasoningResult(nil)) = av('standStill) . --- BUG FIX ---
```

Listing 5. relevant part of MISSCEL-PIRLO interface

The issue can be described as follows: lines 1–2 of Listing 5 show that PIRLO reasons using the meta-execution capabilities of MAUDE (`metaExec`) [21]. This aspect of MAUDE is particularly powerful, as it is possible to write higher-order MAUDE programs that can manipulate and execute MAUDE code. This high expressive power comes at a cost, a high complexity in programming and in tracking and debugging the execution of the code. This might lead to unexpected errors like the deadlock case described. Without entering into details, and treating PIRLO as a black-box system, we found out that, in these two simulations, it happened that PIRLO did not always provide a fluent as a result, e.g., `move(SCID, 0, 0)` for *stand still* or `move(SCID, 1, 0)` for *move up*. Rather, in some cases, the reasoning failed returning terms of form `reasoningResult(nil)`. We can solve the issue by considering these *undecided* cases as *stand still*. We did so by expanding the equations of the operator `reasoner2scel` which encodes back the results given by PIRLO in MISSCEL format. In particular, we only had to add the equation in line 8 of Listing 5. The (old) equation in line 6 will match all and only the non-erroneous cases (further processed by the not-shown equations for operator `translateAction`), while the new one will match all and only the erroneous ones, translating the result in `'standStill`.

After identifying and fixing the causes of the deadlocks, we reran the SMC analyses, collecting the simulation logs. By mining the new logs we obtained the DFG for all robots in Fig. 9. The figure shows that, as expected, the model has no longer deadlocks (the node `deadlock` is no longer included).

5.3 Diff Model

We now present a graphical component automatically computed, the *diff model*, which depicts differences among two DFGs. The role of the diff model is to highlight differences of behavior expressed in the DFGs from two different sets of logs, i.e., the logs for the model with the bug, and those for the fixed model. The DFG shown in Fig. 10 is the diff model between the process model mined before and after the mentioned fix. We can observe two types of edges: solid black edges indicate transitions present in both the original and new fixed versions, while

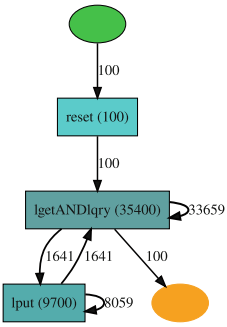

Fig. 9. Process model with informed and normal robots after fixing the model

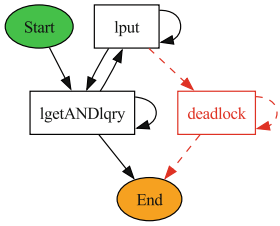

Fig. 10. Diff model of informed robot

dashed red edges, in and out edges of the `deadlock` node, signify transitions no longer present in the new fixed version of the model, as well as the integration of SCEL and Pirlo. In [18], we considered a similar but different notion of diff model, where we use only the logs of one model, and directly compare it with the model as written by the modeler. Such diff model was given in the original model specification language. This approach was dependent on the fact that the formalism considered in [18] had a graphical representation automatically generated from the model. Here, given that SCEL and MISSCEL do not support such an automated graphical representation, we propose a new notion of diff model given directly in PM-terms, i.e., in terms of differences in the DFGs obtained for variants of the model. This can be used, e.g., to compare models at different stages of debugging and validation to assess the provided fixes.

6 Conclusions

We presented a methodology for the programming, modeling, analysis, validation and debugging of collective adaptive systems (CAS) with reasoning capabilities. We integrated SCEL, a language for CAS, the reasoner Pirlo, the statistical analyzer MultiVeStA and process mining techniques (PM). The methodology has been validated on a robotic collision avoidance from the literature. We have shown that the methodology is effective, as it allowed to identify unforeseen issues in material from the literature.

We plan to apply the methodology to more complex scenarios to evaluate its effectiveness in practice. We will consider extending the methodology to scale to scenarios with many components, e.g., by devising techniques where the process of each component is mined, but only the "not conformant" ones are highlighted to the modeler. We will also consider Maude-specific model-reduction techniques [29], or reduction techniques for process algebras [9,10,13,28]. The

paper considers a graphical component that we named diff model. It is a PM-based representation of the differences in the behavior of variants of a model. We plan to extend our approach to provide the diff model in SCEL-based representations, thus not requiring the modeler to understand different formalisms. We will also consider improving the MISSCEL-Pirlo interface using advanced patterns for service-oriented applications [45].

Acknowledgment. The work has been partially supported by Poul Due Jensen Foundation grant no. 883901; S40S Villum Investigator Grant nr. 37819 from Villum Fonden; Fsc regional Tuscan project AISLEA2 J54D23000780005; project SERICS (PE00000014), project SMaRT COnSTRUCT (CUP J53C24001460006), in the context of FAIR (PE0000013, CUP B53C22003630006) under the National Recovery and Resilience Plan (Mission 4, Component 2, Line of Investment 1.3) funded by the European Union - NextGenerationEU.

References

1. van der Aalst, W.M.: Process Mining, 2nd edn. Springer, Cham (2016)
2. Abate, A., Andriushchenko, R., Ceska, M., Kwiatkowska, M.: Adaptive formal approximations of Markov chains. Perform. Eval. **148**, 102207 (2021)
3. Alrahman, Y.A., Nicola, R.D., Loreti, M.: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. **192**, 102428 (2020). https://doi.org/10.1016/J.SCICO.2020.102428
4. Bartoletti, M., Chiang, J.H., Junttila, T., Lluch-Lafuente, A., Mirelli, M., Vandin, A.: Formal analysis of lending pools in decentralized finance. In: Margaria, T., Steffen, B. (eds.) Proceedings of ISoLA 2022. LNCS, vol. 13703, pp. 335–355. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19759-8_21
5. ter Beek, M.H., Legay, A., Lafuente, A.L., Vandin, A.: Quantitative security risk modeling and analysis with RisQFLan. Comput. Secur. **109**, 102381 (2021)
6. ter Beek, M.H., Legay, A., Lluch-Lafuente, A., Vandin, A.: A framework for quantitative modeling and analysis of highly (re)configurable systems. IEEE Trans. Software Eng. **46**(3), 321–345 (2020)
7. Belzner, L.: Action programming in rewriting logic (technical communication). Theory and Practice of Logic Programming, On-line Supplement (2013)
8. Belzner, L., Nicola, R.D., Vandin, A., Wirsing, M.: Reasoning (on) service component ensembles in rewriting logic. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi. LNCS, vol. 8373, pp. 188–211. Springer, Cham (2014). https://doi.org/10.1007/978-3-642-54624-2_10
9. Bernardo, M., Nicola, R.D., Loreti, M.: Revisiting trace and testing equivalences for nondeterministic and probabilistic processes. In: Birkedal, L. (ed.) Proceedings of FOSSACS 2012. LNCS, vol. 7213, pp. 195–209. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-28729-9_13
10. Bernardo, M., Nicola, R.D., Loreti, M.: Relating strong behavioral equivalences for processes with nondeterminism and probabilities. Theor. Comput. Sci. **546**, 63–92 (2014). https://doi.org/10.1016/J.TCS.2014.03.001
11. Berti, A., van Zelst, S., Schuster, D.: PM4Py: a process mining library for python. Softw. Impacts **17**, 100556 (2023). https://doi.org/10.1016/j.simpa.2023.100556

12. Bettini, L., et al.: The Klaim project: theory and practice. In: Priami, C. (ed.) Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop, GC 2003, Rovereto, Italy, 9–14 February 2003, Revised Papers. LNCS, vol. 2874, pp. 88–150. Springer, Cham (2003). https://doi.org/10.1007/978-3-540-40042-4_4
13. Boreale, M., Nicola, R.D.: Testing equivalence for mobile processes. Inf. Comput. **120**(2), 279–303 (1995). https://doi.org/10.1006/INCO.1995.1114
14. Bortolussi, L., et al.: CARMA: collective adaptive resource-sharing Markovian agents. In: Bertrand, N., Tribastone, M. (eds.) Proceedings Thirteenth Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2015, London, UK, 11th–12th April 2015. EPTCS, vol. 194, pp. 16–31 (2015). https://doi.org/10.4204/EPTCS.194.2
15. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Modelling and analyzing adaptive self-assembly strategies with Maude. Sci. Comput. Program. **99**, 75–94 (2015). https://doi.org/10.1016/J.SCICO.2013.11.043
16. Cardelli, L., Perez-Verona, I.C., Tribastone, M., Tschaikowski, M., Vandin, A., Waizmann, T.: Exact maximal reduction of stochastic reaction networks by species lumping. Bioinformatics **37**(15), 2175–2182 (2021)
17. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Comparing chemical reaction networks: a categorical and algorithmic perspective. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 485–494 (2016)
18. Casaluce, R., Burattin, A., Chiaromonte, F., Lluch-Lafuente, A., Vandin, A.: White-box validation of quantitative product lines by statistical model checking and process mining. J. Syst. Softw. **210**, 111983 (2024). https://doi.org/10.1016/J.JSS.2024.111983
19. Casaluce, R., Burattin, A., Chiaromonte, F., Vandin, A.: Process mining meets statistical model checking: towards a novel approach to model validation and enhancement. In: Cabanillas, C., Garmann-Johnsen, N.F., Koschmider, A. (eds.) Business Process Management Workshops - BPM 2022 International Workshops, Münster, Germany, 11–16 September 2022, Revised Selected Papers. LNBIP, vol. 460, pp. 243–256. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-25383-6_18
20. Ciancia, V., Latella, D., Massink, M., Paškauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: ISOLA 2017 (2017)
21. Clavel, M., et al.: All About Maude. LNCS, vol. 4350. Springer, Cham (2007)
22. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A.: A formal approach for the analysis of BPMN collaboration models. JSS **180**, 111007 (2021)
23. Galpin, V., Georgoulas, A., Loreti, M., Vandin, A.: Statistical analysis of CARMA models: an advanced tutorial. In: Johansson, B., Jain, S. (eds.) 2018 Winter Simulation Conference, WSC 2018, Gothenburg, Sweden, 9–12 December 2018, pp. 395–409. IEEE (2018). https://doi.org/10.1109/WSC.2018.8632456
24. Gilmore, S., Reijsbergen, D., Vandin, A.: Transient and steady-state statistical analysis for discrete event simulators. In: IFM, pp. 145–160. Springer, Cham (2017)
25. Gilmore, S., Tribastone, M., Vandin, A.: An analysis pathway for the quantitative evaluation of public transport systems. In: Albert, E., Sekerinski, E. (eds.) IFM 2014. LNCS, vol. 8739, pp. 71–86. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10181-1_5
26. Großmann, G., Kyriakopoulos, C., Bortolussi, L., Wolf, V.: Lumping the approximate master equation for multistate processes on complex networks. In: McIver, A., Horváth, A. (eds.) QEST, vol. 11024, pp. 157–172 (2018)

27. Hasani, R., et al.: Closed-form continuous-depth models. arXiv preprint arXiv:2106.13898 (2021)
28. Iacobelli, G., Tribastone, M., Vandin, A.: Differential bisimulation for a Markovian process algebra. In: Italiano, G., Pighizzini, G., Sannella, D. (eds.) Proceedings of MFCS 2015. LNCS, vol. 9234, pp. 293–306. Springer, Cham (2015). https://doi.org/10.1007/978-3-662-48057-1_23
29. Lluch-Lafuente, A., Meseguer, J., Vandin, A.: State space c-Reductions of concurrent systems in rewriting logic. In: Aoki, T., Taguchi, K. (eds.) Proceedings of ICFEM 2012. LNCS, vol. 7635, pp. 430–446. Springer, Cham (2012). https://doi.org/10.1007/978-3-642-34281-3_30
30. Maus, C., Rybacki, S., Uhrmacher, A.M.: Rule-based multi-level modeling of cell biological systems. BMC Syst. Biol. **5**, 1–20 (2011)
31. Nicola, R.D., Ferrari, G., Pugliese, R.: Locality based Linda: programming with explicit localities. In: Bidoit, M., Dauchet, M. (eds.) TAPSOFT 1997: Theory and Practice of Software Development, 7th International Joint Conference CAAP/-FASE, Lille, France, 14–18 April 1997, Proceedings. LNCS, vol. 1214, pp. 712–726. Springer, Cham (1997). https://doi.org/10.1007/BFB0030636
32. Nicola, R.D., Ferrari, G., Pugliese, R.: KLAIM: a Kernel language for agents interaction and mobility. IEEE Trans. Software Eng. **24**(5), 315–330 (1998). https://doi.org/10.1109/32.685256
33. Nicola, R.D., Gorla, D., Pugliese, R.: On the expressive power of KLAIM-based calculi. Theor. Comput. Sci. **356**(3), 387–421 (2006). https://doi.org/10.1016/J.TCS.2006.02.007
34. Nicola, R.D., et al.: The SCEL language: design, implementation, verification. In: Wirsing, M., Hölzl, M.M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems - The ASCENS Approach. LNCS, vol. 8998, pp. 3–71. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16310-9_1
35. Nicola, R.D., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. ACM Trans. Auton. Adapt. Syst. **9**(2), 7:1–7:29 (2014). https://doi.org/10.1145/2619998
36. Pianini, D., Sebastio, S., Vandin, A.: Distributed statistical analysis of complex systems modeled through a chemical metaphor. In: HPCS, pp. 416–423 (2014)
37. Sebastio, S., Vandin, A.: MultiVeStA: statistical model checking for discrete event simulators. In: Horváth, A., Buchholz, P., Cortellessa, V., Muscariello, L., Squillante, M.S. (eds.) Proceedings of ValueTools 2013, pp. 310–315. ICST/ACM (2013). https://doi.org/10.4108/icst.valuetools.2013.254377
38. Tribastone, M.: Behavioral relations in a process algebra for variants. In: Proceedings of the 18th International Software Product Line Conference, vol. 1, pp. 82–91 (2014)
39. Tschaikowski, M., Tribastone, M.: Spatial fluid limits for stochastic mobile networks. J. Perform. Eval. **109**, 52–76 (2017). https://doi.org/10.1016/j.peva.2016.12.005
40. Vandin, A., ter Beek, M.H., Legay, A., Lluch Lafuente, A.: QFLan: a tool for the quantitative analysis of highly reconfigurable systems. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 329–337. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_19
41. Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F.: MultiVeStA: statistical analysis of economic agent-based models by statistical model checking. In: Bowles, J., Broccia, G., Pellungrini, R. (eds.) From Data to Models and Back - 10th International Symposium, DataMod 2021, Virtual Event, 6–7 December 2021, Revised

Selected Papers. LNCS, vol. 13268, pp. 3–6. Springer, Cham (2021). https://doi.org/10.1007/978-3-031-16011-0_1
42. Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F.: Automated and distributed statistical analysis of economic agent-based models. J. Econ. Dyn. Control **143**, 104458 (2022). https://doi.org/10.1016/j.jedc.2022.104458
43. Vandin, A., Tribastone, M.: Quantitative abstractions for collective adaptive systems. In: Bernardo, M., Nicola, R.D., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, 20–24 June 2016, Advanced Lectures. LNCS, vol. 9700, pp. 202–232. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34096-8_7
44. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166**(July 2017), 1–34 (2006)
45. Wirsing, M., et al.: SENSORIA patterns: augmenting service engineering with formal analysis, transformation and dynamicity. In: Margaria, T., Steffen, B. (eds.) Proceedings of ISoLA 2008. Communications in Computer and Information Science, vol. 17, pp. 170–190. Springer, Cham (2008). https://doi.org/10.1007/978-3-540-88479-8_13